Optimización de rendimiento, justicia y consumo energético en sistemas multicore asimétricos mediante planificación

Doctorado en Ciencias Informáticas
Facultad de Informática - Universidad Nacional de La Plata

Autor: **Adrian Pousa**
III-LIDI, Facultad de Informática, Universidad Nacional de La Plata
apousa@lidi.info.unlp.edu.ar

Fecha de exposición: 11 de Octubre de 2017

Directores:

Ing. Armando De Giusti
III-LIDI, Facultad de Informática, Universidad Nacional de La Plata
degiusti@lidi.info.unlp.edu.ar

Dr. Juan Carlos Sáez Alcaide
Dacya, Universidad Complutense de Madrid, España
jcsaezal@ucm.es

Resumen

Los procesadores multicore asimétricos (AMPs) fueron propuestos como una alternativa de bajo consumo energético a los procesadores multicore convencionales (CMPs) formados por cores idénticos. Los AMPs integran cores rápidos y complejos de alto rendimiento, y cores más simples de bajo consumo. Los AMPs plantean un gran desafío: distribuir eficientemente los ciclos de los distintos tipos de core entre las aplicaciones. Para evitar modificar el código de las aplicaciones, la mayoría de las propuestas se realizan a nivel de sistema operativo incluyendo algoritmos de planificación conscientes de la asimetría. Los algoritmos de planificación para AMPs propuestos hasta el momento intentan optimizar el rendimiento global pero degradan aspectos como la justicia o la eficiencia energética. Asimismo, la mayoría fueron evaluados mediante simuladores o plataformas asimétricas emuladas. El principal objetivo de esta tesis doctoral es superar estas limitaciones diseñando estrategias de planificación conscientes de la justicia y la eficiencia energética alcanzando un rendimiento global aceptable. Implementamos nuestras estrategias en el kernel de un SO real y las evaluamos sobre hardware multicore asimétrico real.

Palabras clave: multicore asimétrico, AMP, planificación, sistema operativo, rendimiento global, justicia, eficiencia energética

1. Introducción

La mayoría de los CMPs (*Chip Multi-Processors*) son simétricos, es decir, están compuestos por cores idénticos. Podemos encontrar CMPs formados por cores complejos (Haswell de Intel o Power8 de IBM) y CMPs formados por cores más simples de consumo reducido (ARM Cortex A9 o Intel Xeon Phi). Los procesadores del primer grupo poseen características microarquitectónicas sofisticadas (ejecución fuera de orden y superescalar) y son adecuados para aplicaciones secuenciales que los usan eficientemente. Los procesadores del segundo grupo son más simples en su microarquitectura y son adecuados para aplicaciones con un elevado paralelismo a nivel de hilo (TLP).

Podemos encontrar CMPs en varias plataformas donde se ejecutan cargas de trabajo muy diversas [13], que presentan distintas demandas y exigen al sistema operativo que optimice objetivos como el rendimiento global, la justicia o la eficiencia energética. Los CMPs presentan una limitación: un modelo particular de CMP puede resultar ideal, en cuanto a rendimiento por *watt*, para un conjunto de aplicaciones pero no para todas. Los AMPs fueron propuestos como alternativa a los CMPs para superar esta limitación [12, 2]. Un AMP está compuesto por dos tipos de core: cores rápidos de alto rendimiento (*big o fast cores*) y cores lentos de bajo consumo (*small o slow cores*). Todos los cores de un AMP poseen el mismo repertorio de instrucciones (ISA - *Instruction-Set Architecture*).[1]

En un AMP es posible utilizar técnicas de *especialización*, es decir garantizar que una aplicación se ejecute en el tipo de core que ofrece la mejor relación entre rendimiento y consumo energético. En general, en un AMP los cores simples son adecuados para la ejecución de aplicaciones paralelas escalables[8]. Por el contrario, los cores complejos son adecuados para la ejecución de aplicaciones secuenciales. En el caso de aplicaciones paralelas que tienen fases secuenciales, es posible utilizar los cores rápidos para acelerar estas fases [1].

1.1. Motivación

A pesar de sus beneficios, los AMPs plantean importantes desafíos [6, 3, 13]. Uno de ellos es distribuir eficientemente los ciclos de los cores rápidos y lentos entre las distintas aplicaciones. Esta responsabilidad recae en el planificador del sistema operativo, de esta forma no es necesario modificar el código de las aplicaciones.

Al inicio de esta tesis doctoral, la mayoría de los algoritmos de planificación propuestos para AMPs tenían como objetivo optimizar el rendimiento global [12, 2, 10]. Para ello, el planificador debe ejecutar en los cores rápidos aquellas aplicaciones que usan estos cores de forma eficiente. La optimización de otros aspectos como la justicia o la eficiencia energética no habían recibido suficiente atención por parte de los investigadores. En esta tesis doctoral demostramos que los planificadores que intentan optimizar sólo el rendimiento global degradan estos otros aspectos.

[1] Utilizar el mismo ISA permite ejecutar el mismo binario en los distintos cores sin tener que recompilar el código para cada uno de ellos.

1.2. Objetivos y principales desafíos

El principal objetivo de esta tesis es diseñar estrategias de planificación a nivel de sistema operativo para optimizar el rendimiento global, la justicia y la eficiencia energética. Para lograr este objetivo fue necesario superar tres grandes desafíos:

- Desarrollamos un *framework* de planificación que facilita la implementación y evaluación de las estrategias de planificación en un entorno realista: un sistema operativo real sobre hardware multicore asimétrico real. Por la complejidad que esto representa muchos investigadores evaluaron sus estrategias de planificación utilizando simuladores [**li10**, 2, 10, 16]. En esta tesis implementamos dichas estrategias sobre un SO real, esto nos permitió identificar ciertas limitaciones de algunas propuestas que no se manifiestan en entornos emulados[4].
- Equipamos al planificador del SO con un mecanismo para estimar en tiempo de ejecución el beneficio relativo (*speedup*) que una aplicación obtiene al usar los distintos tipos de cores en un AMP. Una aplicación puede obtener diferente *speedup* al ejecutarse en cores rápidos con respecto a hacerlo en los cores lentos [2]. Asignar aplicaciones a cores en base a esta diversidad de *speedups* es clave para mejorar el rendimiento global, la justicia o la eficiencia energética en AMPs. En una aplicación secuencial, el *speedup* se conoce como *Speedup Factor SF* y es el ratio del número de instrucciones por segundo (IPS) que el único hilo de la aplicación experimenta en ambos tipos de core ($\frac{IPS_{fast}}{IPS_{slow}}$). En aplicaciones paralelas, el *speedup* se obtiene a partir del *SF* de hilos individuales, el número de hilos de la aplicación y el número de cores rápidos. Medir directamente el *SF* da lugar a imprecisiones y genera *overhead* [15]. Por esta razón, optamos por estimar el *SF*. Para esto, construimos modelos de estimación de *SF* basados en contadores hardware.
- Seleccionamos métricas adecuadas para cuantificar el rendimiento global, la justicia y la eficiencia energética en un AMP. Las métricas que existían al inicio de la tesis estaban definidas para sistemas simétricos y no eran aptas para AMPs. Por este motivo, fue necesario construir nuevas métricas y adaptar métricas definidas previamente para CMPs.

1.3. Contribuciones de la tesis

Las principales contribuciones de la tesis doctoral son las siguientes:

- Construimos un modelo analítico para hallar los planificadores teóricos que optimizan el rendimiento global, la justicia y la eficiencia energética, respectivamente. A partir de éste modelo analizamos la interrelación entre estos tres aspectos y mostramos que no es posible optimizarlos simultaneamente.
- Proponemos los algoritmos de planificación para AMPs: Prop-SP, ACFS, EEF-Driven y ACFS-E. Prop-SP es la primera propuesta de planificación orientada

a justicia en AMPs que tiene en cuenta la diversidad de *speedups* entre aplicaciones. Aunque Prop-SP supera a otras estrategias de planificación, no es capaz de *optimizar* la justicia. Para superar esta limitación creamos ACFS, que maximiza la justicia en AMPs y permite ajustar gradualmente el nivel relativo de justicia y rendimiento global. EEF-Driven es el algoritmo que maximiza la eficiencia energética en AMPs en base al *Energy-Efficiency Factor* (*EEF*). El *EEF* de un hilo es una nueva métrica propuesta en esta tesis que se define como $\frac{SF}{EPI_{fast}}$, donde EPI_{fast} es la energía por instrucción consumida por el hilo en el core rápido. ACFS-E es una variante de ACFS que maximiza la justicia, el rendimiento global y la eficiencia energética en un único algoritmo de planificación. Este planificador permite ajustar el nivel de eficiencia energética o rendimiento global a costa de degradar la justicia de forma gradual.

- Proponemos una metodología para construir modelos precisos de estimación de *SF* y *EEF* basados en el uso de contadores hardware.
- A diferencia de otras propuestas nuestras estrategias ofrecen soporte para acelerar distintos tipos de aplicaciones paralelas.

1.4. Publicaciones derivadas

- J. C. Saez, M. Prieto-Matías, A. Pousa, and A. Fedorova, "Explotación de técnicas de especialización de cores para planificación eficiente en procesadores multicore asimétricos", *Jornadas de paralelismo*, 2011.
- A. Pousa, J. C. Saez, A. D. Giusti, and M. Prieto-Matías, "Evaluation of scheduling algorithms on an asymmetric multicore prototype system", *XX Congreso Argentino de Ciencias de la Computación*, 2014.
- J. C. Saez, A. Pousa, F. Castro, D. Chaver, and M. Prieto Matías, "Exploring the throughput-fairness trade-off on asymmetric multicore systems", *Proc. of Euro-Par 2014: Parallel Processing Workshops - Euro-Par 2014 International Workshops, Part II*, 2014.
- J. C. Saez, A. Pousa, F. Castro, D. Chaver, and M. Prieto-Matías, "ACFS: A completely fair scheduler for asymmetric single-ISA multicore systems", *Proceedings of ACM/SIGAPP Symposium On Applied Computing*, 2015.
- J. C. Saez, A. Pousa, R. Rodriguez-Rodriguez, F. Castro, and M. Prieto-Matías, "PMCtrack: Delivering performance monitoring counter support to the OS scheduler", *The Computer Journal*, 2016.
- A. Pousa, J. C. Saez, F. Castro, D. Chaver, and M. Prieto-Matías, "Towards completely fair scheduling on asymmetric single-ISA multicore processors", *Journal of Parallel and Distributed Computing*, 2017.
- J. C. Saez, A. Pousa, A. D. Giusti, and M. Prieto-Matías, "On the interplay between throughput, fairness and energy efficiency on asymmetric multicore processors", *The Computer Journal*, 2017.

2. Entorno experimental

El entorno experimental utilizado en esta tesis está compuesto por el hardware asimétrico y el framework de planificación para AMPs.

Al inicio de esta tesis no existían plataformas asimétricas. Por esta razón, emulábamos el hardware asimétrico reduciendo la frecuencia de un conjunto de cores en un CMP para obtener los "cores lentos". Durante el desarrollo de la tesis surgieron AMPs con los cuales pudimos experimentar (Intel QuickIA y ARM Juno). Estas plataformas integran cores con diferencias en la frecuencia y en su microarquitectura.

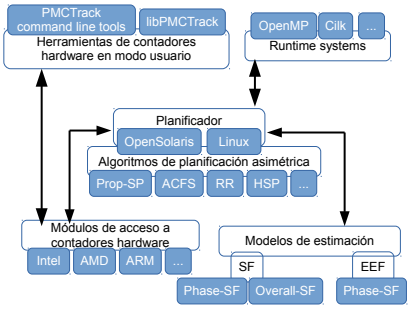


Figura 1: Framework de planificación.

La Figura 1 muestra la estructura del framework de planificación. Su componente central es el planificador del sistema operativo que interactúa con varios componentes que ayudan a la toma de decisiones de los algoritmos de planificación.

3. Métricas

La mayoría de las métricas existentes para cuantificar los distintos objetivos del planificador han sido definidas específicamente para CMPs y no resultan adecuadas para AMPs. Por lo tanto, definimos nuevas métricas y adaptamos métricas existentes.

Para cuantificar el rendimiento global en un AMP utilizamos el *Aggregate Speedup* (ASP), que se define como:

$$ASP = \sum_{i=1}^{n} \frac{CT_{Small,i}}{CT_{Sched,i}} - 1 \quad (1)$$

donde n es el número de aplicaciones en la carga de trabajo, $CT_{Small,i}$ es el tiempo de ejecución de la aplicación i cuando corre sola en el sistema y utiliza solamente los cores lentos, y $CT_{Sched,i}$ es el tiempo de ejecución de la aplicación i ejecutada bajo un planificador determinado junto al resto de aplicaciones de la carga de trabajo.

Para cuantificar la justicia adaptamos la métrica de *injusticia* o *unfairness*[2] propuesta por [14, 5], que se define como:

[2]Un algoritmo de planificación es justo si garantiza que las aplicaciones de una carga de trabajo experimentan la misma degradación del rendimiento (*slowdown*) al ejecutarse de forma simultánea con otras aplicaciones.

$$Unfairness = \frac{max(Slowdown_1, Slowdown_2, \cdots, Slowdown_n)}{min(Slowdown_1, Slowdown_2, \cdots, Slowdown_n)} \quad (2)$$

donde *Slowdown* para una aplicación i se define como $\frac{CT_{Sched,i}}{CT_{Fast,i}}$. $CT_{Fast,i}$ es el tiempo de ejecución de la aplicación i cuando corre sola en el sistema.

Para cuantificar la eficiencia energética en un AMP, utilizamos la métrica *Energy-Delay Product* (EDP)[9, 7], que se define como:

$$EDP = \frac{Energia\_total\_consumida \cdot CT}{Instrucciones\_retiradas\_totales} \quad (3)$$

donde CT es el tiempo de ejecución de la carga de trabajo (tiempo de ejecución de la aplicación más lenta).

4. Algoritmo de planificación Prop-SP

Al inicio de esta tesis, la mayoría de los algoritmos de planificación propuestos para AMPs perseguían maximizar el rendimiento global [12, 2, 10]. Sin embargo, maximizar el rendimiento degradaba la justicia. El objetivo de Prop-SP es ofrecer un buen equilibrio entre estos dos aspectos.

Prop-SP distribuye los ciclos de core rápido entre las aplicaciones utilizando una estrategia basada en créditos inspirada en el planificador Credit Scheduler de Xen sobre CMPs. Cada hilo tiene asociado un contador de *créditos* de core rápido. Aquellos hilos que poseen *créditos* pueden ejecutarse en este tipo de cores. Cuando un hilo se ejecuta en un core rápido, sus créditos se van consumiendo. Cada cierto tiempo, Prop-SP inicia un proceso que otorga créditos de core rápido a las aplicaciones activas teniendo en cuenta sus *speedups*. Cuando un hilo que está ejecutando en un core rápido agota todos sus créditos, el planificador intentará intercambiarlo con otro hilo que esté asignado a un core lento y posea créditos de cores rápidos.

Prop-SP mejora las propuestas de otros autores pero no optimiza la justicia.

5. Relación rendimiento, justicia y eficiencia energética

Realizamos un estudio teórico donde evaluamos la efectividad de diferentes algoritmos de planificación en cuanto al rendimiento global, la justicia y la eficiencia energética. A partir de este estudio analizamos la interrelación entre estos aspectos. Para esto, utilizamos varias cargas de trabajo (Cuadro 2) y las analizamos sobre un AMP compuesto por dos cores rápidos y dos cores lentos.

Cada carga de trabajo está formada por cuatro aplicaciones secuenciales de la suite de benchmarks SPEC CPU (Cuadro 1). Para construir las cargas de trabajo analizamos el comportamiento de varias aplicaciones cuando se ejecutan sobre la placa ARM Juno. Para obtener los valores de IPS y EPI utilizamos los contadores hardware y los registros de energía integrados en esta placa.

Analizamos los valores de las métricas *ASP*, *unfairness* y *EDP* al ejecutar las cargas de trabajo bajo cuatro estrategias de planificación para AMP:

Cuadro 1: Aplicaciones sintéticas

App.	**Benchmark**	**IPS$_{fast}$**	**IPS$_{slow}$**	**SF**	**EPI$_{fast}$**	**EPI$_{slow}$**
A1	art	0.60	0.24	2.47	1.59	1.86
A2	astar	0.58	0.31	1.86	1.40	1.10
A3	bzip2	1.49	0.73	2.02	0.61	0.45
A4	equake	0.80	0.26	3.07	1.31	1.45
A5	galgel	1.30	0.41	3.16	0.82	0.91
A6	gamess	2.01	0.69	2.91	0.51	0.49
A7	gobmk	1.09	0.61	1.79	0.75	0.54
A8	gzip	1.07	0.63	1.70	0.78	0.56
A9	h264ref	1.83	0.93	1.96	0.51	0.37
A10	hmmer	2.80	1.04	2.69	0.42	0.36
A11	mcf	0.18	0.09	2.02	3.98	4.44
A12	mgrid	1.28	0.59	2.17	0.85	0.65
A13	perlbench	1.42	0.71	2.01	0.60	0.47
A14	perlbmk	1.77	0.78	2.27	0.54	0.41
A15	povray	1.15	0.53	2.19	0.85	0.66
A16	soplex	0.52	0.20	2.53	1.68	1.86
A17	swim	0.25	0.11	2.24	3.11	3.34
A18	vortex	1.73	0.72	2.41	0.56	0.45
A19	wupwise	1.63	0.64	2.56	0.66	0.54

Cuadro 2: Cargas de trabajo

Carga de trabajo	Aplicaciones
W1	A5,A4,A6,A10
W2	A16,A17,A13,A9
W3	A16,A1,A18,A14
W4	A5,A4,A10,A15
W5	A5,A4,A6,A12
W6	A1,A12,A3,A13
W7	A1,A17,A7,A8
W8	A15,A11,A13,A2
W9	A4,A11,A3,A8
W10	A10,A19,A16,A9

- HSP: planificador que optimiza el rendimiento asignando a cores rápidos aquellas aplicaciones de la carga de trabajo con mayor *speedup*.
- *Opt-Unf*: planificador teórico que optimiza la justicia (asegura el óptimo *unfairness* - valor más bajo - para el máximo valor de *ASP* alcanzable).
- *Opt-EDP*: planificador teórico que optimiza la eficiencia energética (asegura el óptimo *EDP* -valor más bajo- para el máximo valor de *ASP* alcanzable).
- EEF-Driven: nuestro planificador que asigna a cores rápidos las aplicaciones que alcanzan el valor más alto de *EEF* (las restantes se asignan a cores lentos).

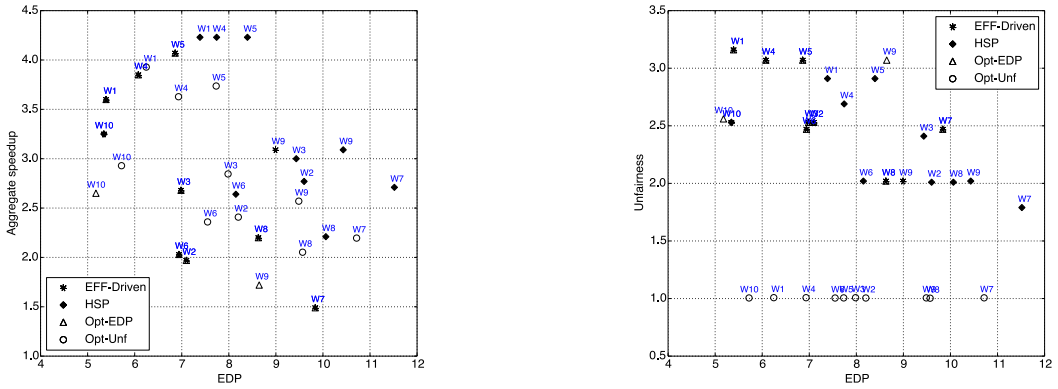
Las Figuras 2a y 2b muestran la relación entre rendimiento global (ASP) y eficiencia energética (EDP), y justicia ($unfairness$) y eficiencia energética, respectivamente. HSP alcanza los valores más altos de ASP a expensas de degradar la eficiencia energética. *Opt-EDP* alcanza el menor valor de *EDP* pero degrada significativamente el rendimiento global. Asimismo, tanto HSP como *Opt-EDP* son inherentemente injustos. Por otro lado, *Opt-Unf* alcanza los mejores valores de $unfairness$ pero degrada tanto el rendimiento global como la eficiencia energética.

La principal conclusión que obtenemos de estos resultados es que la justicia, el rendimiento global y la eficiencia energética son objetivos contrapuestos, ya que cualquier intento de optimizar una de las métricas lleva a degradar las otras.

Opt-Unf y *Opt-EDP* no pueden ser implementados en el kernel de un SO, debido a que son algoritmos de orden exponencial. Observamos que EEF-Driven aproxima a *Opt-EDP*, ya que ambas realizan la misma distribución de ciclos de core rápido.

6. Algoritmo de planificación ACFS

ACFS intenta aproximar el planificador óptimo de justicia (*Opt-Unf*) mediante el seguimiento del progreso realizado por los distintos hilos de la carga de trabajo durante su ejecución. Para seguir el progreso de las distintas aplicaciones, el planificador



(a) EDP vs.ASP (b) EDP vs. Unfairness

Figura 2: ASP, EDP y Unfairness para las cargas de trabajo del Cuadro 2 bajo distintos planificadores. Los mejores resultados en a) se encuentran próximos a la esquina superior izquierda y en b) se encuentran próximos a la esquina inferior izquierda.

asocia a cada hilo un contador (*amp_vruntime*). Cuando un hilo se ejecuta durante un *tick* de reloj en un tipo de core determinado, ACFS incrementa el contador del hilo en $\Delta amp\_vruntime$, que se calcula como sigue:

$$\Delta amp\_vruntime = \frac{100 \cdot W_{def}}{S_{core} \cdot W_t} \tag{4}$$

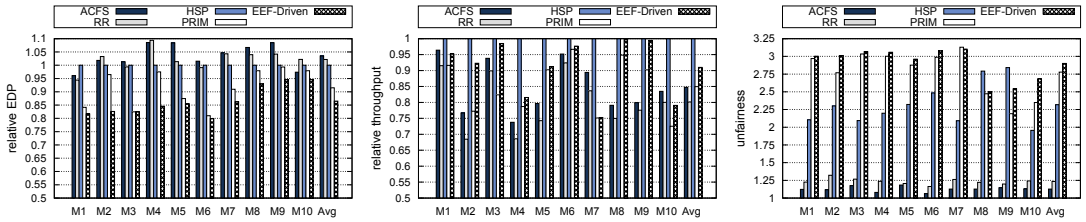
donde W_{def} es el peso de las aplicaciones con la prioridad por defecto, S_{core} es la degradación en rendimiento (*slowdown*) experimentada por la aplicación y W_t es el peso del hilo (prioridad especificada por el usuario).

Cuando un hilo se ejecuta en un core rápido, $S_{core} = 1$ (no hay degradación del rendimiento). Si el hilo se ejecuta en un core lento $S_{core} = Speedup$.

ACFS aproxima el *Speedup* en tiempo de ejecución teniendo en cuenta el *SF* del hilo y el número de hilos activos de la aplicación. A su vez, el *SF* del hilo se estima alimentando un modelo de estimación con valores de métricas de rendimiento obtenidos para el hilo mediante contadores hardware.

Para garantizar justicia, el planificador debe lograr que las aplicaciones realicen el mismo progreso. Para esto, ACFS garantiza que el valor de los contadores de progreso de los hilos sea lo más cercano posible, realizando migraciones de hilos entre los diferentes tipos de core cada cierto tiempo. ACFS efectúa intercambios de hilos cuando detecta que la diferencia entre los contadores de progreso (*amp_vruntime*) de estos supera un cierto umbral.

ACFS está equipado con un parámetro de configuración, *Unfairness Factor* o *UF*, que permite incrementar gradualmente el rendimiento global del sistema en escenarios donde los requisitos de justicia son menos estrictos.



(a) EDP relativo (b) ASP relativo (c) Unfairness

Figura 3: Resultados para las cargas de trabajo bajo la configuración 2F-4S

7. Análisis experimental

Comparamos EEF-Driven y ACFS con otros planificadores para AMP: HSP, PRIM [16] (orientado a mejorar la eficiencia energética) y RR (orientado a proporcionar justicia). Utilizamos la placa de desarrollo ARM Juno que integra 2 cores rápidos y 4 cores lentos (2F-4S). Evaluamos 10 cargas de trabajo (M1-M10), cada una compuesta por 6 aplicaciones de la suite de benchmarks SPEC CPU.

La Figura 3 muestra los valores de *EDP*, *ASP* y *Unfairness* obtenidos. Los valores de EDP y ASP están normalizados con respecto a los resultados de HSP, y las cargas de trabajo están ordenadas por el *EDP* relativo dado por EEF-Driven.

Los resultados experimentales muestran tendencias similares al estudio teórico: optimizar una métrica puede llevar a degradar las otras. HSP alcanza los mejores valores de *ASP* pero sufre de degradación en *EDP*. EEF-Driven obtiene los mejores valores de *EDP* pero sufre de degradación rendimiento. Ambos planificadores sufren de degradación en la justicia.

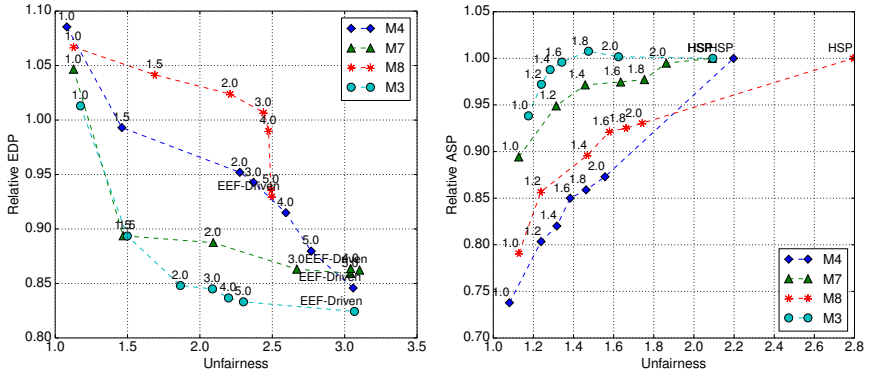
ACFS obtiene mejoras en justicia pero degrada el *EDP* y el *ASP*. ACFS supera a RR en rendimiento global y justicia. Esto se debe a que ACFS tiene en cuenta el *speedup* de las aplicaciones al tomar decisiones de planificación, a diferencia de RR.

EEF-Driven supera a PRIM en varios aspectos. En particular, PRIM hace intercambio de hilos de forma aleatoria, lo que provoca asignaciones de hilos a cores subóptimas y esto lo lleva a obtener valores de *EDP* peores a los esperados.

8. Algoritmo de planificación ACFS-E

Las estrategias de planificación evaluadas anteriormente ofrecen un compromiso fijo entre justicia, rendimiento global y eficiencia energética. Dado que estos aspectos no se pueden optimizar simultaneamente diseñamos ACFS-E. Este planificador, variante de ACFS, está equipado con dos parámetros de configuración: *EDP_factor* y *Unfairness_factor* (UF). ACFS-E permite modificar la relación eficiencia energética-justicia mediante *EDP_factor*, o rendimiento global-justicia mediante *UF*. Cuando estos parámetros se establecen a sus valores por defecto, el algoritmo se comporta como la implementación base del algoritmo ACFS, que maximiza la justicia.

Analizamos cuatro cargas de trabajo bajo ACFS-E. Los resultados se muestran en las Figuras 4a y 4b. La Figura 4a muestra el impacto sobre la justicia y la efi-



(a) Unfairness vs. EDP (b) Unfairness vs. ASP

Figura 4: a) EDP Factor y b) Unfairness Factor.

ciencia energética al variar *EDP_factor*. En general, observamos que valores altos de *EDP_factor* tienden a reducir el *EDP* degradando la justicia. Así, ACFS-E se aproxima a EEF-Driven. La Figura 4b muestra el impacto sobre la justicia y el rendimiento global al variar *UF*. En general, observamos que valores altos de *UF* mejoran el rendimiento global degradando la justicia. Así, ACFS-E se aproxima a HSP.

Conclusiones

El principal objetivo de esta tesis doctoral ha sido diseñar estrategias de planificación para AMPs sobre un sistema operativo real, conscientes de la justicia y de la eficiencia energética, garantizando un rendimiento global aceptable. Con este objetivo, analizamos la relación entre estos tres aspectos y concluimos que son objetivos contrapuestos y no pueden ser optimizados simultáneamente en un AMP.

Este análisis fue clave para el diseño de nuestras estrategias de planificación: Prop-SP, ACFS, EEF-Driven y ACFS-E. El objetivo de Prop-SP es ofrecer un buen equilibrio entre justicia y rendimiento global. Sin embargo, este algoritmo no es capaz de optimizar la justicia en un AMP. Para superar esta limitación diseñamos ACFS, un algoritmo que aproxima el comportamiento del planificador teórico que optimiza la justicia en un AMP. Además, permite ajustar gradualmente el nivel relativo de justicia y rendimiento global. EEF-Driven aproxima el planificador teórico que optimiza la eficiencia energética en AMPs. ACFS-E es una variante de ACFS que constituye la primera estrategia de planificación para AMPs que puede configurarse para optimizar el rendimiento global, la justicia y la eficiencia energética individualmente, con un único algoritmo de planificación. Asimismo, ACFS-E permite al usuario ajustar el compromiso rendimiento-justicia o energía-justicia.

En el análisis experimental, mostramos que nuestros algoritmos superan estrategias propuestas por otros autores.

Referencias

[1] Murali Annavaram, Ed Grochowski y John Shen. "Mitigating Amdahls Law through EPI Throttling". En: Proc. of ISCA 05 (2005), págs. 298-309.

[2] Michela Becchi y Patrick Crowley. "Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures". En: Proc. of CF 06 (2006), págs. 29-40.

[3] N. Chitlur y col. "QuickIA: Exploring heterogeneous architectures on real prototypes". En: HPCA 12 (2012), págs. 1-8.

[4] Kenzo Van Craeynest y col. "Fairness-aware scheduling on single-ISA heterogeneous multi-cores". En: PACT 13 (2013), págs. 177-187.

[5] Eiman Ebrahimi y col. "Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems". En: ASPLOS 10 (2010).

[6] Matt Gillespie. "Preparing for The Second Stage of Multi-Core HW: Asymmetric (Heterogeneous) Cores". En: *Intel White Paper* (2008).

[7] R. Gonzalez y M. Horowitz. "Energy dissipation in general purpose microprocessors." En: Solid-State Circuits, IEEE Journal (1996), págs. 1277-1284.

[8] M. D. Hill y M. R. Marty. "Amdahls Law in the Multicore Era". En: *IEEE Computer* (2008), págs. 33-38.

[9] M. Horowitz, T. Indermaur y R. Gonzalez. "Low-power digital design." En: Digest of Technical Papers., IEEE Symposium (1994), págs. 8-11.

[10] David Koufaty, Dheeraj Reddy y Scott Hahn. "Bias Scheduling in Heterogeneous Multi-core Architectures". En: Proc. of Eurosys 10 (2010).

[11] Farkas Kumar y Jouppi. "Single-ISA Heterogeneous Multi-Core Architectures: the Potential for Processor Power Reduction". En: Proc. of MICRO 36, 03 (2003).

[12] Rakesh Kumar y col. "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance". En: ISCA 04 (2004), págs. 64-.

[13] Sparsh Mittal. "A Survey Of Techniques for Architecting and Managing Asymmetric Multicore Processors". En: *ACM Computing Surveys* (2016).

[14] Onur Mutlu y Thomas Moscibroda. "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors". En: Proc. of MICRO '07 (2007). DOI: `http://dx.doi.org/10.1109/MICRO.2007.40`.

[15] Daniel Shelepov y col. "HASS a Scheduler for Heterogeneous Multicore Systems". En: *ACM SIGOPS OSR* 43.2 (2009).

[16] Y. Zhang y col. "Cross-architecture prediction based scheduling for energy efficient execution on single-isa heterogeneous chip-multiprocessors." En: Microprocess. Microsyst.,39 (2015), págs. 271-285.